Projet de navigateur HTML – Partie I

Résumé

Il s'agit d'implanter la fonction d'affichage d'un navigateur HTML (le reste du programme est fourni dans un squelette disponible sur le site du cours).

Modalité d'évaluation du projet :

- Soutenance : pendant l'un des 3 derniers TP de l'année : présentation des fonctionnalités sur des exemple préparés + questions (10-15 minutes au total).
- Rendu intermédiaire obligatoire (affichage simple, avec ou sans mise en page, avec ou sans gestion des balises) : 24 mai à minuit . Par mail (adresse fournie en TP ou sur le site du cours).
- Rendu final : 28 juin minuit. Par mail (adresse fournie en TP ou sur le site du cours).

1 Introduction

Une page Web est un texte écrit dans le format informatique *Hypertext Markup Language* (HTML) qui donne au navigateur le texte à afficher ainsi que la structure générale de la mise en page : titres et paragraphes, listes, tableaux. La manière dont la structure influe sur l'affichage dépend du navigateur et des options choisies par l'utilisateur. Ainsi le fichier HTML suivant sera affiché approximativement comme à la figure 1.

```
<html>
<h1>Essai:</h1> ce <i>mot</i> est en <font color="red">italique</font>. <p> Nouveau
<b> paragraphe </b>, les
lignes sont <u>découpées</u>.</p>
<p><b>L'imbrication <i>des <u>balises</u></i></b> est possible.</p>
<p> les <a href="autrefichier.html">ancres</a> sont permises.</p>
</html>
```

FIGURE 1 – Exemple d'affichage d'un fichier HTML

Le but de ce projet est de programmer un petit navigateur HTML capable d'afficher correctement un texte comportant des balises HTML (décrites plus bas). Il sera également possible de *cliquer* sur un lien pour afficher un autre fichier ou une autre partie du même fichier.

Certaines balises acceptent des *attributs*, comme la balise `<font>` qui accepte l'attribut `color`. Ces attributs sont en général optionnels. Une page web commence nécessairement par la balise `<html>` et se termine par `</html>`. Consultez l'annexe B et surtout le web pour plus de précisions.

2 Ce qui vous est demandé

Vous devez implanter *la fonction d'affichage* `display` dans le programme `navig` fourni. Le programme prend en argument un nom de fichier (local, il n'est bien entendu pas demandé d'accéder au réseau). Il devra alors afficher le fichier dans la fenêtre graphique en interprétant les balises HTML correctement, avec une mise en page correcte (les paragraphes trop longs devront être affichés sur plusieurs lignes). Une fois l'affichage effectué, le programme attendra que l'utilisateur effectue une action et réagira en fonction de cette action. Les différentes actions sont les suivantes :

Clique de souris sur un lien	⇝ affiche un autre fichier ;
Appui sur la touche "q"	⇝ quitte le programme ;
Appui sur "←"	⇝ revient au fichier précédent ;
(optionnel) Appui sur "page suivante" ou "page précédente"	⇝ affiche la page précédente/suivante ;
Autre action	⇝ ne fait rien.

Une liste des balises que le navigateur devra savoir afficher est donnée en annexe B. Il en existe d'autre dont certaines sont reconnues par le parseur qui vous est fourni (voir plus bas).

La fonction d'affichage aura pour effet d'afficher le contenu du fichier dans la fenêtre. Les deux difficultés principales sont les suivantes :

- Gérer l'imbrication des balises. Par exemple le mot `balises` dans l'exemple ci-dessus est affiché en gras, en italique et en souligné car il se trouve à l'intérieur de ces trois balises simultanément.
- Couper les lignes afin de faire tenir le texte dans la fenêtre.

Pour résoudre ces deux difficulté, on utilise deux procédés :

- On représente le contenu HTML sous la forme d'un *arbre*, qui reflète les imbrications de balises. Le type de cet arbre (`html_tree`) est fourni dans le squelette du programme, voir section 3 pour plus d'explication. Le mécanisme de lecture du fichier et de construction de l'arbre *est déjà fourni également*.
- On défini un type `display_state` contenant toutes les informations d'affichage (position courante dans la fenêtre, numéro de la ligne courante, couleur courante, etc) nécessaires pendant l'affichage.

La fonction d'affichage `display` sera donc récursive, elle prendra en argument un arbre HTML et *un argument de type* `display_state` permettant de savoir ou et comment afficher les prochains mots. Lors des appels récursifs sur les sous-arbres de l'arbre HTML, on passera le `display_state` reflétant l'effet de la balise rencontrée. On prendra soin de ne *pas* utiliser de variable globale pour l'affichage, car cela ne permet pas de gérer l'imbrication des balises.

Comme la position d'affichage est modifiée par la fonction d'affichage, celle-ci doit retourner un `display_state` reflétant ce changement.

Finalement, on a le profil suivant pour la fonction d`display` :

```
let rec display (harbre:html_tree) (etat_affichage:display_state) : display_state =
...
```

3 Ce qui vous est fourni

Vous disposez des modules suivants, ils sont documentés dans le répertoire `doc` du projet. Pour la visualiser lancez un navigateur html sur le fichier `doc/index.html`.

- `Html_tree` (fichier `html_tree.ml(i)`), dans lequel la fonction `build_html_tree s` permet de lire le fichier nommé `s` et de construire un *arbre*. Le type des arbres est aussi défini dans le module `Html_tree`. Il reflète la structure de la page HTML. C'est le type que vous allez manipuler pour afficher la page HTML.
  Ce module fait appel aux deux modules `Lex_navig` et `Parse_navig` *que vous n'avez pas à manipuler ni à comprendre*.
- `Graph_utils` (fichier `graph_utils.ml(i)`) qui contient des fonctionnalités d'affichages basées sur le module `Graphics` de la librairie de OCaml. Si nécessaire ajoutez des fonctionnalité à ce module (en n'oubliant pas de mettre dans le `.mli` la signature des fonctions à exporter).
- `Navig` (fichier `navig.ml`) qui sera votre module principal, et qui contient des exemples d'utilisation des autres modules. En principe vous implanterez vos fonctions dans ce fichier.

Vous disposez également d'un fichier `Makefile` qui permet de compiler automatiquement votre projet. Pour compiler votre projet, tapez `make`. Si vous ajoutez des fichiers au projet, il vous faut maintenir la liste des fichiers qui se trouve à la ligne `SRCMLFILES=...` et `SRCMLIFILE=...` du fichier `Makefile`, puis taper `make depend`.

Une documentation des librairies standard de OCaml se trouve à l'adresse suivante (section `The standard library`) : http://caml.inria.fr/pub/docs/manual-ocaml/index.html

Le type des arbres HTML est fourni dans le module `Html_tree`. Comme les balises peuvent être *imbriquées*, la structure de donnée la plus pratique et le plus naturelle pour représenter un contenu HTML est l'*arbre*.

```
type attribut =
   Href of string
   | Hcolor of string
     (** À COMPLÉTER SI NÉCESSAIRE *)
   | Size of int

type html_tree =
  | Empty
  | Html of attribut list * html_tree list
  | Head of attribut list * html_tree list
  | Body of attribut list * html_tree list
  | P of attribut list * html_tree list
  | B of attribut list * html_tree list
  | I of attribut list * html_tree list
  | U of attribut list * html_tree list
  | Font of attribut list * html_tree list
  | A of attribut list * html_tree list
  | H1 of attribut list * html_tree list
  | H2 of attribut list * html_tree list
  | H3 of attribut list * html_tree list
  | H4 of attribut list * html_tree list
  | H5 of attribut list * html_tree list
  | Center of attribut list * html_tree list
  | Words of string list
```

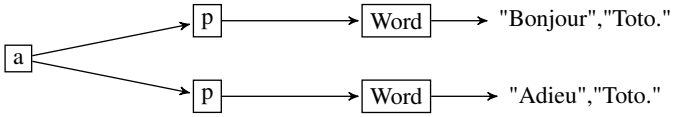
Le type `html_tree` représente une page web. Il s'agit d'un arbre à branchement quelconque : chaque noeud à un nombre arbitraire de fils, stockés dans une *liste*. Il y a également une liste d'attributs (éventuellement vide) sur chaque noeud (sauf les noeuds de type [Word]). Par exemple :

```
<A href="fichier.html">
 <p>Bonjour Toto.</p>
 <p>Aurevoir Toto.</p>
</A>
```

Sera représenté par l'arbre suivant :

```
A([Href("fichier.html")] ,
  [ P([] , [ Words (["Bonjour" ; "Toto." ]) ]) ;
    P([] , [ Words (["Adieu" ; "Toto." ]) ]) ])
```

lui-même schématisé comme suit (on omet les attributs ici) :

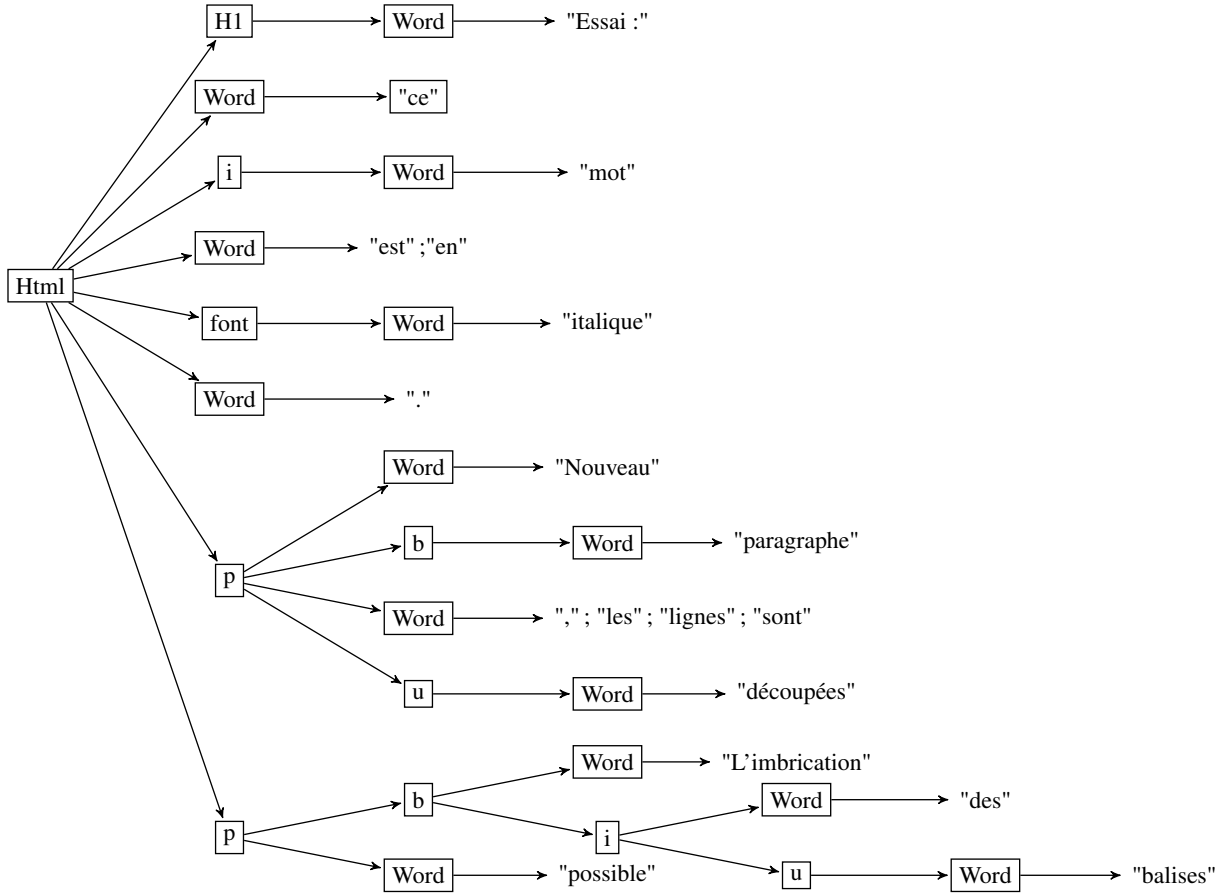


Le schéma de l'arbre HTML correspondant à l'exemple de l'introduction est donné en annexe A.

A Exemple d'arbre HTML

Dans cet exemple, on voit que la structure reflète l'imbrication des balise. Par exemple les balises <b>, <i> et <u> sont imbriquées dans le dernier paragraphe.

```
<html>
<h1>Essai:</h1> ce <i>mot</i> est en <font color="red">italique</font>. <p> Nouveau
<b> paragraphe </b>, les
lignes sont <u>découpées</u>.</p>
<p><b>L'imbrication <i>des <u>balises</u></i></b> est possible.</p>
<p> les <a href="autrefichier.html">ancres</a> sont permises.</p>
</html>
```



B Liste des balises HTML traitées

`<p>texte</p>`	Texte dans un paragraphe
`<b>texte</b>`	**Texte en gras**
`<i>texte</i>`	*Texte en italique*
`<u>texte</u>`	Texte souligné
`<a href="lien.html"> texte </a>`	Insère un lien hypertexte
`<font color="#cc0000"> texte</font>`	Affiche le texte dans la couleur choisie
`<font size="2"> texte</font>`	Change la taille du texte
`<body text="#cc0000"> texte</body>`	Annonce la partie principale de la page, afficher avec la couleur par défaut donnée dans `text`.
` `	Retour à la ligne forcé
`<center>texte</center>`	Centrage du texte
`<em>texte</em>`	mise en avant du texte (en général italique)
`<h1>texte</h1>`	Taille de texte 1 (Grande)
`<h2>texte</h2>`	Taille de texte 2
`<h3>texte</h3>`	Taille de texte 3
`<h4>texte</h4>`	Taille de texte 4
`<h5>texte</h5>`	Taille de texte 5 (Petite)
`<head>texte</head>`	Annonce l'en-tête du fichier, ne pas afficher
`<html>texte</html>`	Annonce le début d'une page, obligatoire
`<title>texte</title>`	Texte est le tire de la page, ne pas afficher